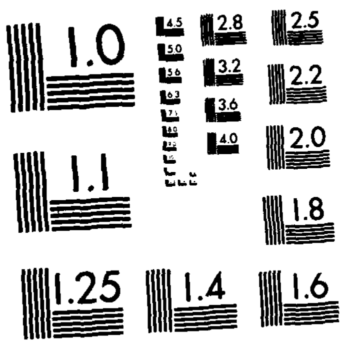
AD-A125 561 HIGH-PERFORMANCE BANDED AND PROFILE EQUATION SOLVERS FOR THE CRAY-1 I THE.(U) MICHIGAN UNIV ANN ARBOR SYSTEMS ENGINEERING LAB D A CALAHAN 01 FEB 82 SEL-160 1/1

UNCLASSIFIED AFOSR-TR-83-0078 AFOSR-80-0158 F/G 12/1 NL

END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Report No. 160

High-Performance Banded and Profile Equation Solvers for the CRAY-1

I. The Unsymmetric Case

D.A. CALAHAN

February 1, 1982

Sponsored by Directorate of Mathematical and Information Sciences, Air Force Office of Scientific Research, under Grant No. ~~80-0158~~ – AFOSR-80-0158

Systems Engineering Laboratory

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR- 83-0078
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): HIGH-PERFORMANCE BANDED AND PROFILE EQUATION SOLVERS FOR THE CRAY-1: I. THE UNSYMMETRIC CASE
5. TYPE OF REPORT & PERIOD COVERED: Interim
6. PERFORMING ORG. REPORT NUMBER: SEL # 160
7. AUTHOR(s): D. A. Calahan
8. CONTRACT OR GRANT NUMBER(s): AFOSR 80-0158
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Michigan, Dept. of Elec. & Computer Engring. Ann Arbor, MI, 48109
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304/A3
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research(NM) Bolling AFB, Washington, DC, 20332
12. REPORT DATE: February 1, 1982
13. NUMBER OF PAGES: 49
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Sparse matrices
Parallel processing
Vector processing
Linear algebra

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes algorithms, performance, applications, and user information associated with two equation-solving codes for the CRAY-1: (1) Solution of a single banded matrix equation, unsymmetric in value but symmetric in structure; (2) Solution of a single profile matrix equation, unsymmetric in valus and in structure. Both solvers assume that the matrix is main-memory resident.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

High Performance Banded and Profile Equation-Solvers for the CRAY-1

I. The Unsymmetric Case

D. A. Calahan

Systems Engineering Laboratory
University of Michigan
Ann Arbor, Michigan 48109

February 1, 1982

A

SEL Report #160

Sponsored by the Directorate of Mathematical and Information Sciences, Air Force Office of Scientific Research, under Grant 80-0158

ABSTRACT

This report describes algorithms, performance, applications, and user information associated with two equation-solving codes for the CRAY-1:

(1) Solution of a single banded matrix equation, unsymmetric in value but symmetric in structure;

(2) Solution of a single profile matrix equation, unsymmetric in value and in structure.

Both solvers assume that the matrix is main-memory resident. The former partitions the matrix internally to achieve high performance. The latter requires a user-supplied blocking of the LU structure, an inconvenience compensated by higher performance in solution of finite difference and a finite element grids.

These codes are available as part of a library of CAL-coded equation-solvers [13].

PREFACE

The mathematical software described herein is the result of experimental research on vector algorithms for the direct solution of 2-D finite difference and finite element grids. The latter code represents what is thought to be the best compromise between vectorizability, sparsity exploitation, and user convenience for such problems for the CRAY-1.

REVISION NOTICE

Pages 18f were revised on August 15, 1982, to include discussion of blocking algorithms.

TABLE OF CONTENTS

	PAGE
I. Banded & General Sparsity Solution.	
A. Introduction.	1
B. General Sparse Solvers.	2
C. Block-Oriented Solvers.	2
D. Band-Oriented Solvers	3
E. Report Summary.	4
II. Solution of a Single Banded Matrix Equation	5
A. Memory-resident Banded Systems.	5
B. Algorithms and Implementation	8
C. Performance	13
D. Software Description.	17
III. Solution of a Block Profile Matrix Equation	18
A. Motivation for Block Profile Solution	18
B. Blocking Model and Irregular Grids.	19
C. Implementation.	33
D. Performance	34
E. Software Description.	39
F. Example Problems.	41
References.	

I. Banded and General Sparsity Solution

A. Introduction

Studies of the direct solution of 2-D finite element grids have tended to take one of two directions, depending on the nature of the sparsity.

1. Band-related methods. Solvers that recognize sparsity principally outside a band around the diagonal are termed band-related solvers. This bandwidth may vary - envelope, skyline, or profile solvers - and may be implicit - as in frontal methods where a matrix is never fully formed. These solvers account for an easy majority of direct solution methods in production codes.
2. General sparsity methods. The early work of George [5] indicated for the first time the possibility of reduced operation counts using codes that permit an arbitrary sparsity pattern, termed general sparsity methods. This spawned a number of research studies on such methods.

It now appears that general sparsity methods, at least when applied to matrices sized to reside in the memory of current processors, are difficult to vectorize [2][6][7]. Rather, only "large-scale" sparsity patterns can be vectorized to achieve a high performance. This conclusion is documented in the following section.

B. General Sparse Solvers

These solvers accept arbitrary sparsity structure in column- or row-ordered form and an associated pivot order. After a preprocessing step to generate the LU fill structure, multiple numerical solutions may then be performed [8]. Early studies of vectorization of these scalar algorithms sought to maintain the same input data structure and carry out the same number of floating point operations as their scalar predecessors [9]. The imposition of these two constraints was justifiable to establish a performance standard that may be achieved without alteration of common user data structures and without introduction of the issue of trading off floating point computation for higher vector performance. By defining vectors within dense regions of the LU structure, an average vector length ($\bar{\ell}$) was defined [6]. It was possible to establish the vectorizability of the solution of finite element grids exploiting such density [6]. Because $\bar{\ell}$ increases monotonically with grid size, sufficiently large 2-D grids can always be satisfactorily vectorized.

C. Block-Oriented Solvers

Unfortunately, as nxn 2-D grids increase, operation counts increase at least as $O(n^3)$ and direct solution methods become less attractive than iterative techniques. However, vector processors have made the solution of 3-D and time-dependent 2-D problems feasible; in such cases, repeated direct solution of a moderate-sized 2-D grid often appears as a computational kernel in a global iterative strategy. For such moderate-sized grids, one cannot depend on randomly-produced density to achieve long vectors.

The first concession to vectorization must be to abandon the traditional general sparsity input data structures. Rather, the user* must assist in the vectorization process by detecting either repeated [10] or dense [2] substructures in the matrix.

For the CRAY-1, vectorized block reduction can be performed rather efficiently [2]. Such blocks are easily detected at the local level when many (r) unknowns are associated with each node and/or a finite element has a large number of associated unknowns [4]; all operations can then be visioned as occurring between rxr full matrices. The overall execution rate (in MFLOPS) can often be estimated from the rate of the multiply-accumulate kernel that accounts for the major part of the computation in the reduction of each pivot block. Since no extra computation is performed in such block-oriented elimination, the solution time is inversely related to this execution rate.

A study of the solution time with such solvers on the CRAY-1 shows that only part of the speedup over scalar solvers is due to vectorization. A significant advantage also accrues from the need to address only <u>blocks</u> rather than single elements of the sparse matrix, since this processor is known to be slow in the indirect addressing mode (gather/scatter) associated with linked list processing.

D. Band-Oriented Solvers

Even the best coding on the CRAY-1 -acknowledged to have superior short-vector performance - cannot achieve above 20-30 MFLOPS with fewer than five unknowns/node. To achieve rates in the range of 100 MFLOPS requires the significantly larger dense substructures that

---

*Algorithms to prepare the structure for vectorization could, of course, be considered.

are associated with inter-nodal coupling. Such coupling is usually along a line or adjacent lines (multi-line coupling) that yields a banded matrix structure. Indeed, the natural (grid-row) ordering of irregular 2-D grids yields a step banded structure, shown in Figure 5(b). If one can justify performing somewhat extra computation in the gaps between such steps, the entire solution can be performed with a locally-banded solution mode.

It is the conclusion of the experiences to be reported that such block profile matrix solution offers the best performance compromise between bandsolvers that assume an absolutely regular sparsity structure and general sparse solvers that permit random structures. The study of such a "large-scale sparsity" methods is the goal of this research.

E. Report Summary

To establish a standard for performance comparison it was essential to first code an efficient bandsolver in assembly language (CAL). This was a non-trivial task; the memory-hierarchial CRAY-1 architectur required a partitioned solution process. The first part of this report describes the algorithms, implementation, and performance of this software. Its speedup over Fortran implementations makes this useful in its own right.

The block profile solution is then discussed and is liberally documented with examples to give insight into the class of problems for which it yields improved performance over the above standard.

II. Solution of a Single Banded Matrix Equation

A. Memory-resident Banded Systems

In reference [1], Jordan has presented an algorithm for solving a banded set of equations on the CRAY-1 and gave the performance of associated CAL software. Unfortunately, the 64-length vector limitation of the CRAY-1 resulted in the code being applicable to matrices with half-bandwidths $b \leq 64$. This part of the report describes the design and performance of software that does not have this restriction.

It can be argued that very large banded sets of equations cannot be solved with the entire matrix resident in main memory, an assumption of the code to be described. Nonetheless, an intermediate range of matrix sizes beyond the above bandwidth restriction can be stored in even ½-megaword configurations, and yet solved in fractions of a second. With up to 4-megaword systems in the offing, it is likely that the majority of banded matrices will be beyond the capability of Jordan's code.

Because most banded matrices arise from the solution of partial differential equations, consider the banded matrix produced by applying the 5-point finite difference formula to the 2D grid of Figure 1, where

n_s is the shorter grid dimension
n_ℓ is the longer grid dimension
u is the number of unknowns per grid point
$k_\ell = n_s/n_\ell$ is the ratio of grid dimensions

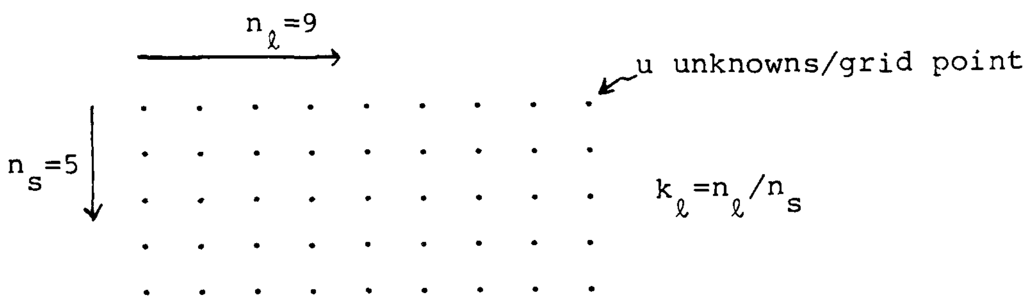


Figure 1. Definition of grid descriptors

u / s/k_ℓ	1	2	4	8
4	126:126	159:79	109:49	247:30
2	100:100	127:63	155:38	199:24
1	79: 79	101:50	123:30	151:18
½	63: 63	79:39	99:24	119:14
¼	50: 50	63:31	75:18	95:11
⅛	39: 39	49:24	59:14	71: 8

Table 1. $b_{max}:n_{smax}$ as a function of matrix storage (megawords) and number of unknowns/grid point; examples below dashed line have $b_{max} \leq 64$.

The matrix is of order $n=un_s n_\ell$, with a half-bandwidth b=[illegible] -1. The matrix storage in compressed form (Figure 2) is

$$\begin{aligned} s &= (2u(n_s+1)-1)(un_s n_\ell) \\ &= (2u(n_s+1)-1)(uk_\ell n_s^{\,2}) \end{aligned} \tag{1}$$

For $n_s >> 1$,

$$s \approx 2k_\ell u^2 n_s^{\,3} \tag{2}$$

Asymptotically,

$$n_s = \left(\frac{s}{2k_\ell u^2}\right)^{1/3}$$

and the half-bandwidth is

$$\begin{aligned} b &= un_s \\ &= .794u^{1/3}(s/k_\ell)^{1/3} \end{aligned} \tag{3}$$

With $b \leq 64$, it is clear from (3) that grids with a constant k_ℓ will be more impacted by this restriction as u increases. The precise nature of this restriction is indicated in Table 1. For example, a one-megaword processor with a grid dimension ratio $k_\ell=2$ will accommodate the matrix only when $u \approx 1$; yet the associated equations can be solved in only 330 msec at 100 MFLOPS, a representative execution rate. Even a square 50 x 50 grid with u=4, executing on a four-megaword system, can be solved in only 8 seconds, and yet generates a half-bandwidth of 203, well beyond the 64-length limitation.

In conclusion, it appears that many 2D grids producing matrices with bandwidths greater than 64 can be solved in reasonable time with direct methods on present and near-term memory configurations. The more general program to be described can be expected to extend the usefulness of many application codes that are based on direct methods to such problems.

B. Algorithms and Implementation

1. Problem statement and solution

It is desired to solve a banded set of equations

$$AX = B$$

where A is an n x n unsymmetric matrix of half-bandwidth m, and is sufficiently well-conditioned that pivoting is not required.

The solution is performed by partitioned LU factorization of A in one subroutine (BANFAC) and by partitioned forward and back substitution in a second subroutine (BANSOL).

2. Storage options

In different applications disciplines, it is customary to store the matrix in one of two compressed formats. In row storage format, illustrated in Figure 2(b), the diagonals are stored in rows of a (2m+1) x n array; in column storage format, the diagonals are stored in columns of a n x (2m+1) array. The voids are assumed to be zero-valued.

3. Inner loop algorithm

The algorithm of Jordan's inner loop code is adopted for this more general code, although the coding is somewhat different.

Jordan's LU factorization uses the following accumulation for the jth columns.

$$U^{(k+1)}_{r+1:s,j} = u^{(k)}_{r+1:s,j} - u_{rj}L_{r+1:s,r} \tag{5}$$

$$L^{(k+1)}_{j+1:t,j} = L^{(k)}_{j+1:t,j} - u_{rj}L_{j+1:t,r} \tag{6}$$

$$L_{j+1:t,j} = L^{(j-r)}_{j+1:t,j}/u_{jj} \tag{7}$$

where $s=\min(r+m,j)$, $t=\min(r+m,n)$, $r=r_0,\ldots,j-1$, $k=r-r_0$, $r=\max(j-m,1)$, and $U_{a:b,j}(L_{a:b,j})$ represents a vector of components $u_{ij}(\ell_{ij})$ with $a\le i\le b$.

Since the calculation of $u_{r_0+k,j}$ is completed after the kth step and since the component u_{ij} (or l_{ij}) is unaffected by the accumulation until $i\le r+m$, the vector length remains m until $r+m>n$. After each kth step, the first vector element $u_{r_0+k,j}$ or $\ell_{r_0+k,l}$ is removed by a vector shift and a new final element $u_{r+m+1,j}$ or $\ell_{r+m+1,j}$ added at the end of vector.

Once removed from the vector, the completed element immediately becomes a scalar multiplier u_{rj} in (5) and (6). This removal process creates a delay in Jordan's code, and a subsequent small loss in asymptotic MFLOP rate. The delay is removed in the new code by precalculating the first vector element in scalar mode. The resulting accumulation loop has a timing formula

$$T_\ell = 17 + VL \qquad VL \geq 30 \tag{8}$$

where VL is the vector length; the associated execution rate is 126 MFLOPS for VL=64. For VL<30, the timing is approximately a constant 47 clocks.

4. Partitioning

To extend Jordan's code beyond half-bandwidths of 64 while maintaining the high execution rates associated with vector accumulation loops such as the above, the matrix must be partitioned into 64 x 64 blocks, noted by Jordan [1] for full matrices and Calahan [2] for block sparse matrices. In [2], the partitioning of banded matrices was performed into square blocks and "bandedge" blocks, with a degradation in processing the latter. In the present code, the partitioning is performed into diagonal blocks, as illustrated in Figure 3. Loop 1 is a single vector operation; loop 2 is the inner loop of vector operations which terminate after 64 accumulations into the jth column. In loop 3, the next 64 elements of the same 64 columns are accumulated into the jth column. The scalar pre-calculation noted above for the inner loop is unnecessary for the blocks non-adjacent to the diagonal, and a somewhat more efficient accumulation loop is utilized. Loop 3 continues until the bandedge is encountered. Loop 4 then advances the accumulation to the next column of 64 blocks, as shown in Figure 3.

When the bottom of the matrix is encountered in the processing of a block, one is faced with either testing for this condition in the inner loop -- and thus adding a fixed inner loop overhead -- and then reducing the vector length, or simply carrying out the additional floating point calculations. It happens that the matrix storage format permits the latter during factorization, so that this procedure

$$\begin{array}{cccccc}
a_{11} & a_{12} & a_{13} & & & \\
a_{21} & a_{22} & a_{23} & a_{24} & & \\
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & \\
 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\
 & & a_{53} & a_{54} & a_{55} & a_{56} \\
 & & & a_{64} & a_{65} & a_{66}
\end{array}$$

Logical storage (m=2,n=6)

$$\begin{array}{cccccc}
0 & 0 & a_{13} & a_{24} & a_{35} & a_{46} \\
0 & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\
a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\
a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & 0 \\
a_{31} & a_{42} & a_{53} & a_{64} & 0 & 0
\end{array}$$

Row storage (5 x 6 array)

$$\begin{array}{ccccc}
0 & 0 & a_{11} & a_{12} & a_{13} \\
0 & a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\
a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\
a_{53} & a_{54} & a_{55} & a_{56} & 0 \\
a_{64} & a_{65} & a_{66} & 0 & 0
\end{array}$$

Column storage (6 x 5 array)

Figure 2. Band matrix storage formats.

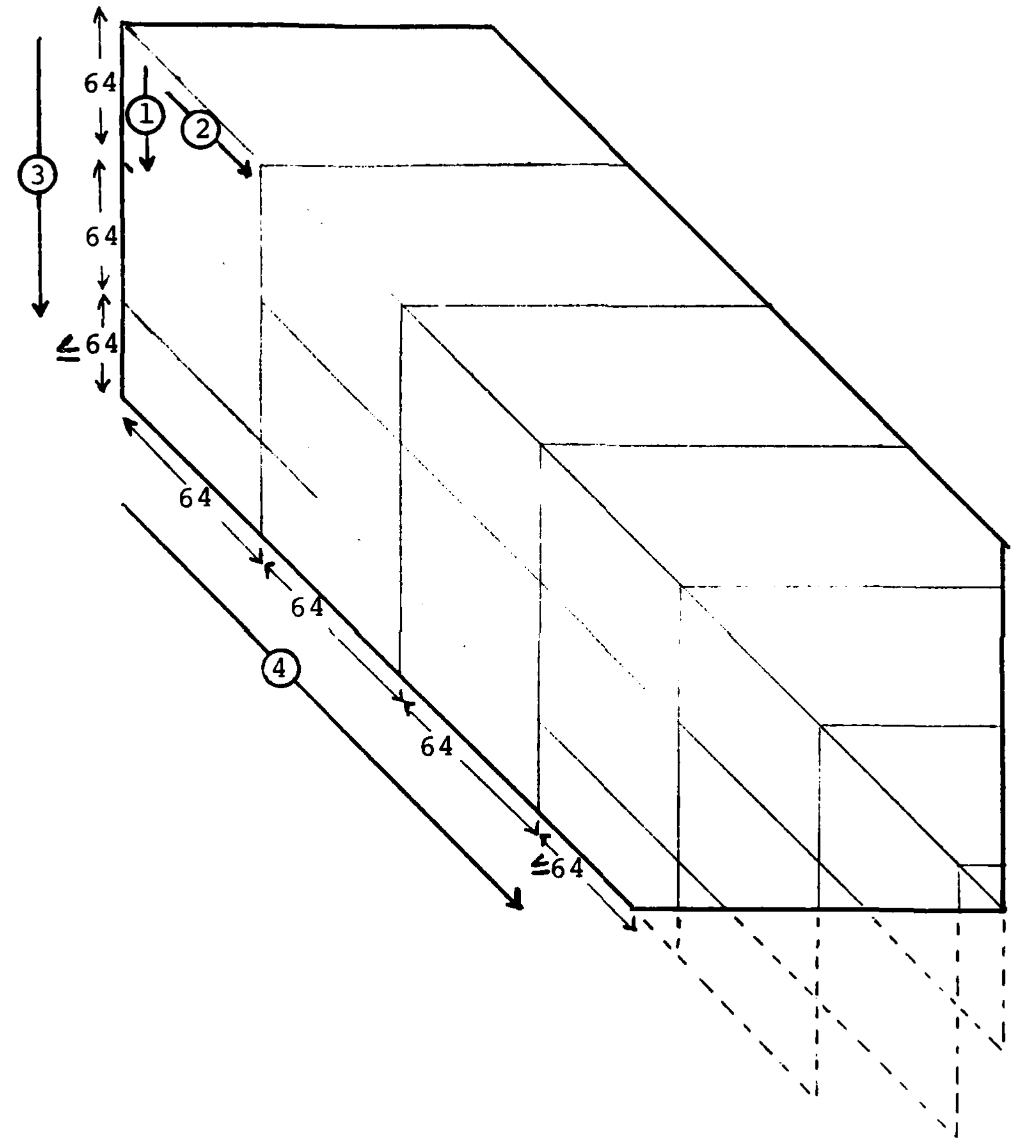


Figure 3. Illustrative partitioned matrix

was chosen. The extra block processing is indicated by dashed lines in Figure 3. For large n/m ratios, the faction of extra computation is bounded by m/3n. During the substitution steps, the ratio is bounded by m/2n; also, the right hand side is relocated on entry and exit, to provide requisite void storage space.

C. Performance

In comparison with the timing of Jordan's unpartitioned code, the partitioned version incurs somewhat more overhead to implement the partitioning and to allow row- or column-ordered storage. However, use of a simulator [3] has allowed a somewhat more careful attention to inner loop timing (see below). This effort is deemed worthwhile, since the direct solution of large dense matrix equations inevitably dominates the equation formulation, with the result that the total execution time tends to be closely related to the performance of this inner loop.

Table 2 gives the performance statistics* of the partitioned versus the unpartitioned code. As indicated, the specialized inner loop coding more than compensates for the outer loop overhead. Also the execution rates for even small bandwidths easily exceed those of scalar processors (in the order of 1-5 MFLOPS).

The effects of partitioning are shown in Figure 4. For bandwidths less than 65, no partitioning is necessary and all vector lengths are equal to the bandwidth. For $65 \le m \le 128$, the average vector length is approximately m/2. A resultant sharp drop in execution rate occurs for m=65. For $129 \le m \le 192$, the average length is 2m/3, so that the decrease for m=129 is less severe.

Figure 4 also presents the measured rates of this software versus the Fortran-coded LINPACK bandsolvers SGBFA and SGBSL available at CRI (a slower version of these codes on the UCS system in 9/81 was also tested). Because these codes allow pivoting (which commonly accounts for 25-30% of the solution time in CAL for large matrices), this comparison is somewhat unfair to these Fortran codes.

It is perhaps more intuitive to relate the performance directly to the grid from which banded matrices are normally derived. Table 3 gives the computing times and execution rates for a number of cases up to the storage limits of a megaword machine. The execution rates uniformly are in the range of 100 MFLOPS for large problems, ranging up to the limit 118 MFLOPS for a 64 x 64 grid with one unknown per grid point.

---

*All performance results in this report are derived from runs on the megaword CRAY-1 configuration at United Computing Systems, Inc.

Half Bandwidth	Time : Rate (ms : MFLOPS*)	Improvement
Factorization		
8	1.88:18.2	2.1
16	2.97:43.6	1.9
32	5.66:86.0	1.7
64	16.0:110.	1.3
96	35.9:99.2	--
128	54.6:103.	--
Solution		
8	.315:26.4	1.3
16	.316:51.0	1.3
32	.358:86.1	1.3
64	.565:102	1.1
96	.921:87.1	--
128	1.14:86.2	--

*Extra computation ignored in operation count (see text)

Table 2. Timing performance of partitioned code for 256 equations, and comparison with Jordan's original code [1].

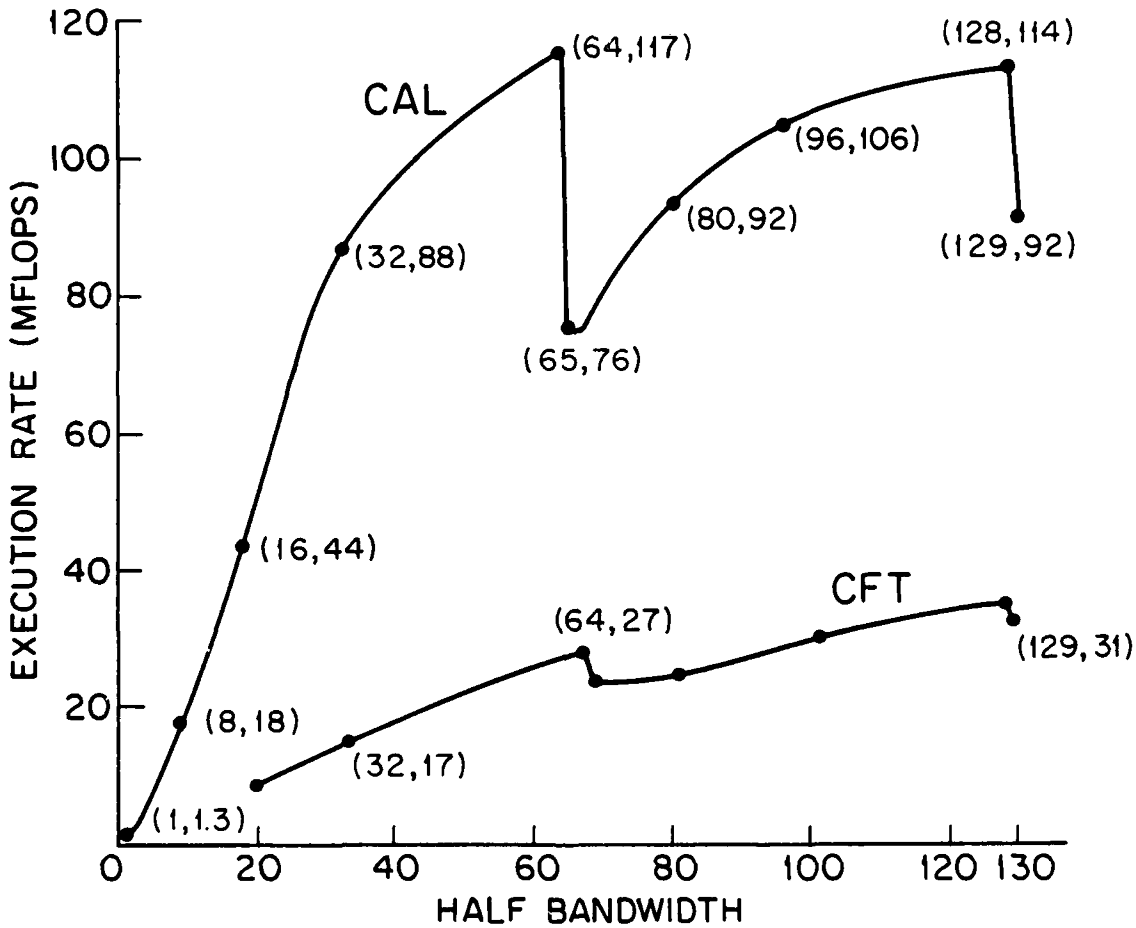


Figure 4. Performance of banded partitioned (CAL) and LINPACK (CFT) timings for solving 1024 equations; Note the LINPACK codes include pivoting.

u / n_s	1	2	4
8	.457:17.5	1.52:45.4	6.50:89.0
16	2.97:43.6	12.3:88.2	115:76.8
20			221:96.7
24		51.1:106.	413:107.
28			718:114.
32	23.6:88.4	224:76.2	
40		445:93.3	
64	283:118		

(a) Factorization (BANFAC)

u / n_s	1	2	4
8	.085:28.1	.162:50.1	.376:89.1
16	.315:51.0	.720:91.4	3.49:76.2
20			5.43:95.6
24		2.07:107	8.55:105.
28			12.7:112.
32	1.40:93.0	7.05:74.6	
40		11.1:92.7	
64	8.84:118.		

(b) Solution (BANSOL)

Table 3. Time (msec): execution rate (MFLOPS) as a function of square grid size (n_s) and number of unknowns per grid point (u).

D. Software Description

1. Calling sequence

Factorization

```
CALL BANFAC(N,M,A,NDIAG,NDROW)
```

Substitution

```
CALL BANSOL(N,M,A,NDIAG,NDROW,B)
```

where

N is the number of equations

M is the half-bandwidth, not including diagonal

A is the compressed band matrix array

NDIAG is the diagonal addressing increment

NDROW is the row addressing increment

B is the right hand side and the solution array

2. Explanation

1. NDIAG is the storage addressing increment between logical matrix positions (i,j) and (i+1,j+1), i.e. between successive diagonal elements. For row storage, NDIAG≥2*M+1; for column storage, NDIAG=1.

2. NDROW is the storage addressing increment between logical matrix positions (i,j) and (i+1,j), i.e. between successive column elements. For row storage, NDROW=1; for column storage, NDROW=-(N-1).

3. Restrictions

1. Storage must be zero-valued beyond active compressed banded storage (Figure 2).

2. Storage for B must be at least N+2*M.

3. |NDROW| should not be a multiple of 8, to avoid bank conflicts.

III. Solution of a Block Profile Matrix Equation

A. Motivation for Profile Solution

Profile matrices tend to arise in two ways.

1. The "natural" column-by-column (or row-by-row) reduction of nodes in a 2-D grid produces a banded matrix for rectangular grids only. For grids with irregular external boundaries, a variable bandwidth (profile) matrix results.
2. Floating point computation can be approximately halved in solution of a large grid defined by a 5-point operator, by first reducing unrelated nodes. This step requires insignificant computation for large grids; however, it halves the matrix size and leaves the bandwidth unchanged. If the unrelated nodes are eliminated along alternate diagonals (termed AD or D4 ordering [11]) and then the remaining nodes numbered along diagonals, a profile matrix results. The LU factors of such a matrix are illustrated in Figure 5(b) for the 8x12 grid of Figure 5(a). By exploiting this profile, floating point operations can be reduced by another factor of 2 for a large square grid.

In the examples of this report, it is assumed that the alternate diagonals have been eliminated in the first step and only a profile matrix remains. On the CRAY-1, the vectorization of this first step is highly dependent on the regularity of matrix storage, since considerable data movement but little computation is involved. With random storage and scalar operations from FORTRAN, a rate less than 1 MFLOP may be achieved. With a patterned storage, from CAL a rate of 70 MFLOPS has been observed. In the first case, the significance of the time associated with this first step can be ignored only with very large grids.

Storage permitting, the profile matrix could be solved by the bandsolver previously described. If the profile solver were to operate at the same execution rate as the bandsolver (an optimistic assumption), then on a square grid the solution time would be halved. This factor of 2 is therefore an upper limit on speed-up of profile over banded solution. Note that this is far less than the 3-5 speedup factor which CAL achieves vis-a-vis CFT (Figure 4).

B. Block Profile Solution

1. Blocking Rationale

The vectorization of the solution is further assisted by "regularization" of the matrix structure into two-dimensional blocks, for two reasons.

(1) Fewer symbolic descriptors relating to block size and storage locations are necessary to describe a block than to describe the same matrix elements either individually or as a collection of 1-dimensional dense columns or rows (as in [9]). The processing of these descriptors can add significant overhead to numeric processing, especially when processing small blocks on a vector processor.

(2) The high speed CRAY-1 vector register set has a single critical data path to main memory. The utilization of this path can be reduced by performing matrix-matrix or

```
xx - 01 - xx - 03 - xx - 07 - xx - 13 - xx - 21 - xx - 29
      |    |    |    |    |    |    |    |    |    |    |
02 - xx - 04 - xx - 08 - xx - 14 - xx - 22 - xx - 30 - xx
 |    |    |    |    |    |    |    |    |    |    |    |
xx - 05 - xx - 09 - xx - 15 - xx - 23 - xx - 31 - xx - 37
 |    |    |    |    |    |    |    |    |    |    |    |
06 - xx - 10 - xx - 16 - xx - 24 - xx - 32 - xx - 38 - xx
 |    |    |    |    |    |    |    |    |    |    |    |
xx - 11 - xx - 17 - xx - 25 - xx - 33 - xx - 39 - xx - 43
 |    |    |    |    |    |    |    |    |    |    |    |
12 - xx - 18 - xx - 26 - xx - 34 - xx - 40 - xx - 44 - xx
 |    |    |    |    |    |    |    |    |    |    |    |
xx - 19 - xx - 27 - xx - 35 - xx - 41 - xx - 45 - xx - 47
 |    |    |    |    |    |    |    |    |    |    |    |
20 - xx - 28 - xx - 36 - xx - 42 - xx - 46 - xx - 48 - xx
```

Figure 5(a). AD-ordered 8x12 grid; xx represents nodes eliminated in pre-reduction step. Matrix shown in Figure 5(b).

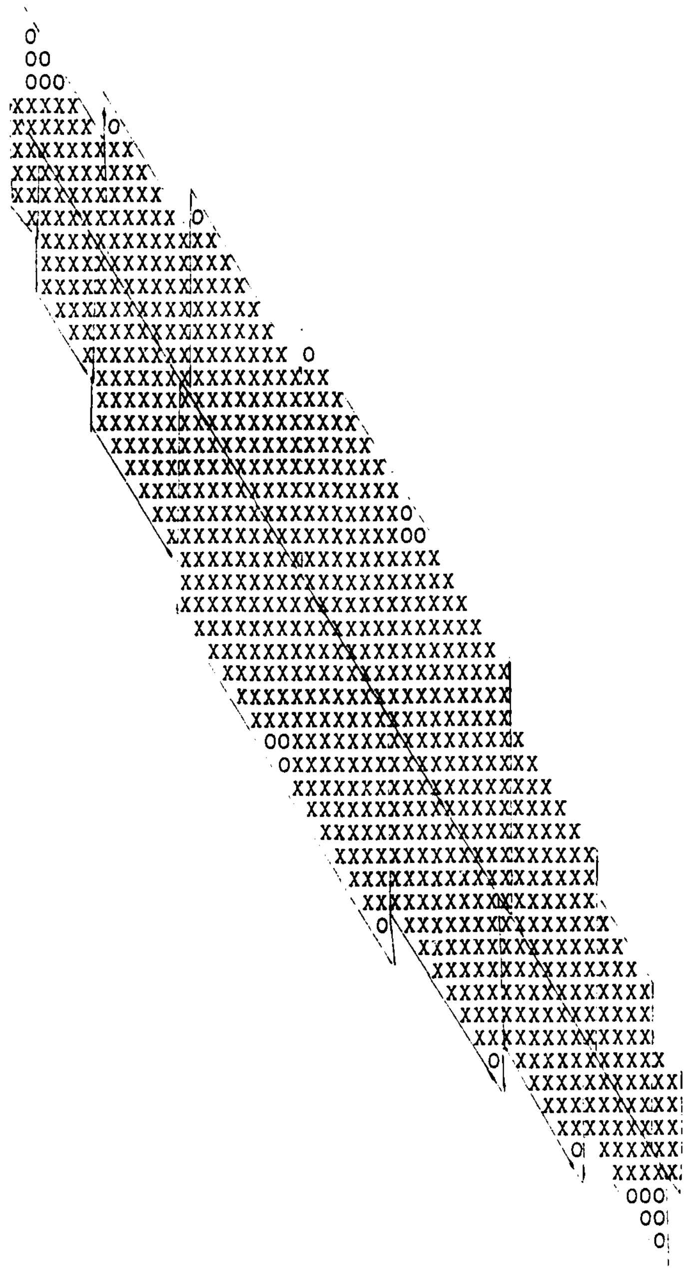


Figure 5(b) Blocked profile matrix associated with 8x12 grid with u = 1; 0 - zero-valued position inserted for blocking.

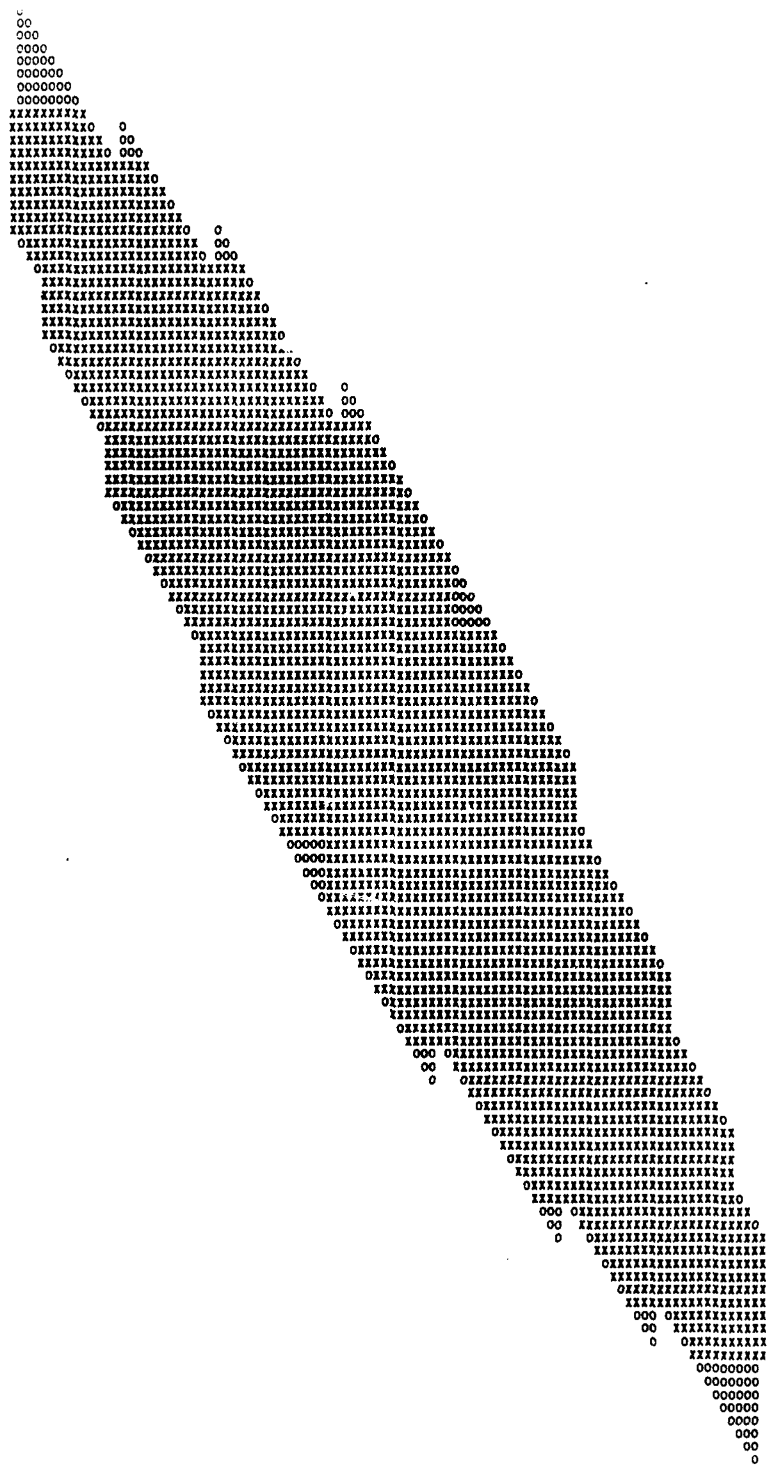
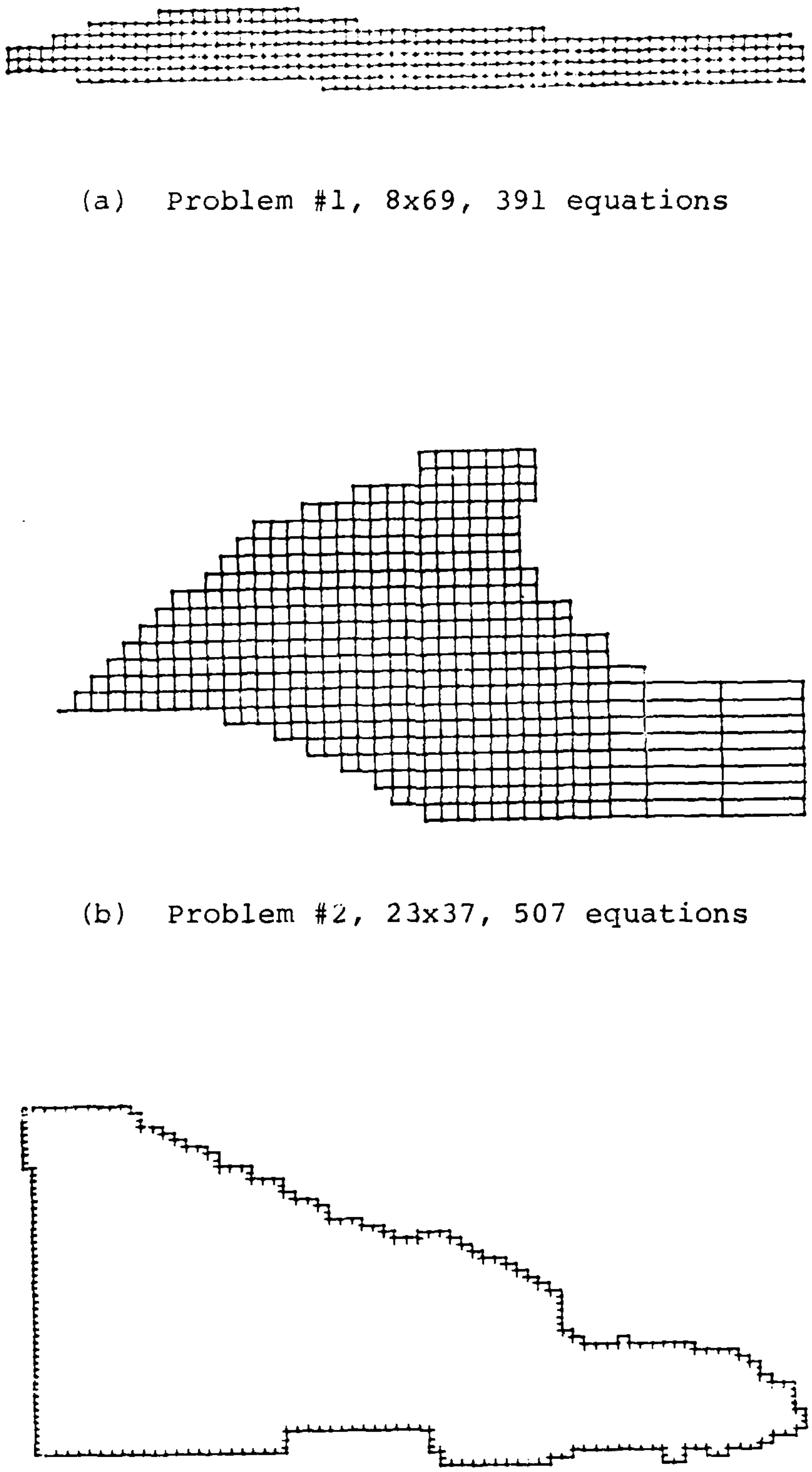


Figure 5(c). Blocked profile matrix associated with 8x12 grid, with u = 2; 0 - zero-valued position inserted for blocking.

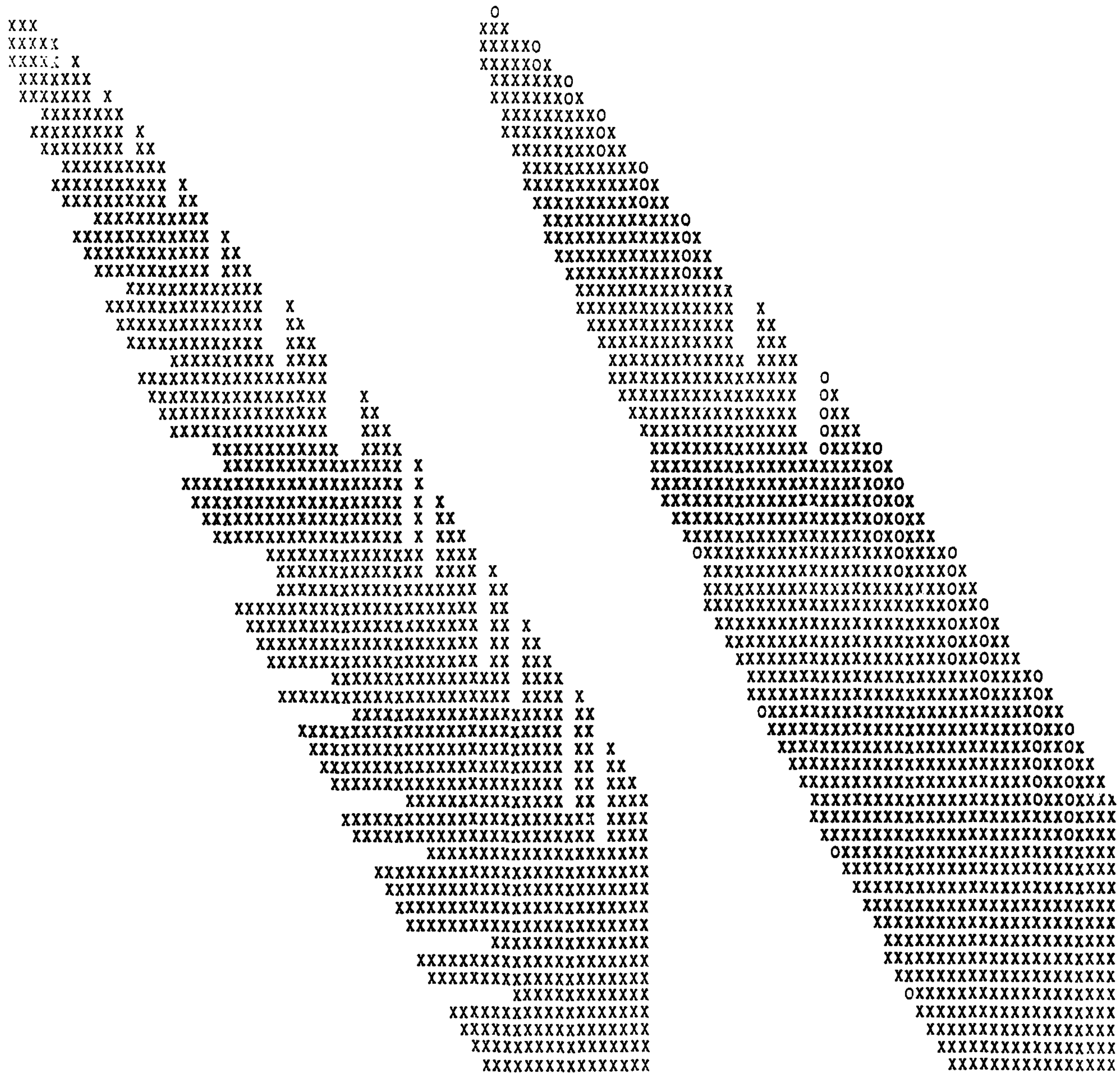


(a) Problem #1, 8x69, 391 equations

(b) Problem #2, 23x37, 507 equations

(c) Problem #3, 55x72, 2323 equations

Figure 6. Irregular grids



(a) Original LU factors

(b) After column-ordered elimination and blocking

Figure 7. Northwest 61x61 partition of 507-equation LU factors; with AD ordering after elimination of alternate nodes; 0-zero-valued positions added for blocking.

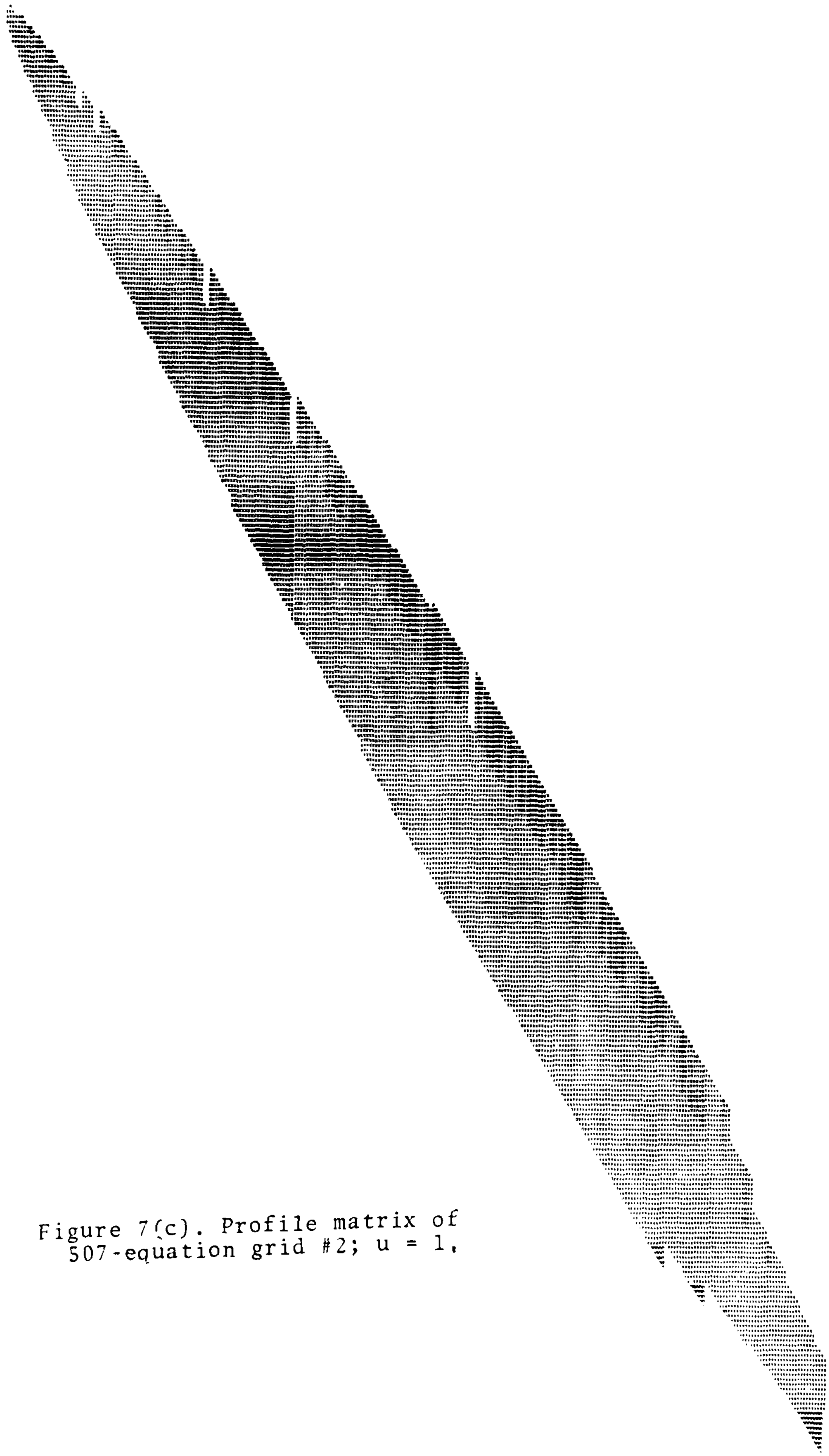


Figure 7(c). Profile matrix of 507-equation grid #2; u = 1.

matrix-vector rather than vector-vector operations. The identification of matrix (or block) structures is thus essential to achieving highest execution rates.

2. Blocking Attributes

To gain insight into desirable blocking attributes, two classes of problems are studied in this report.

(a) The model grid of Figure 1, reduced to a profile matrix by D4 ordering.

(b) A set of three irregular grids of Figure 6, taken from [14]. Unrelated nodes are pre-reduced, but in the natural ordering of the grid rather than along diagonals. The profile of the resulting matrix is then similar to the profile of the matrix resulting from natural ordering of the entire grid; however, the number of equations is approximately halved.

The structure of the LU matrix from an AD-ordered model grid has been shown in Figure 5(b). For the irregular grid of Figure 6(b), a submatrix is shown in Figure 7(a). In both cases, the natural matrix boundaries occur not in rectangular blocks but in blocks whose boundaries are parallel to the diagonal. This characteristic is consistent with the partitioning strategy of the band-solver; this suggests that the accumulation kernels of the latter may be used in this case.

Unfortunately, these accumulation kernels require that a column of L be considered dense from the diagonal to the largest row number (contrast, an inner-product accumulation). As a result, for the irregular grid of Figure 6(b), the columns of the L factor are extended to the bandedge, as shown in Figure 7(b). The resulting extra computation associated with extra non-zeros will be evaluated later empirically. Non-zeros need not be added to the U matrix for this reason.

A second accumulation kernel characteristic is the assumption that successive columns being accumulated begin and end one row number apart. This observation sets the primary requirement of the blocking algorithm for L and U, i.e., the identification of two-dimensional submatrices, each bounded above and below by diagonals parallel to the main diagonal and by columns on each side.

In this blocking process, one can judiciously add non-zeros to complete blocks, as in Figure 3 near the southeast corner of the banded matrix. This becomes the critical part of the blocking algorithm and will be considered in detail later. This addition of non-zeros in L will suffice for blocking of the factorization and forward substitution. However, if the back substitution is to proceed efficiently, blocks in U must also be completed as above.

Matrices blocked in L and U by the algorithm below are illustrated in Figures 5(b) and 7(b) for model and irregular girds, respectively.

3. Blocking Algorithm

An optimal blocking algorithm must have as its goal minimization of the factorization (or solution) time by reducing the number of blocks without adding excessive computation or storage. Development of this algorithm would proceed by

(a) coding the block factorization and solution algorithm,

(b) developing a detailed timing model for each of the major loops in the code, and

(c) solving for the location of block separators that minimize the execution time.

This problem can be phased as a nonlinear programming problem. The nonlinearity arises from the possibility of overlap between scalar and vector operations: scalar computation may hide a concurrent short vector operation, or it may itself be hidden by a long vector operation. The dependence of computation time on vector length is therefore a nonlinear one. Even without this dependence, the arbitrary insertion of block separators is an integer programming problem and so is beyond reasonable solution time for large profile systems.

A local - and thus suboptimal - minimization algorithm has been developed that focuses on the local irregularities of the profile. It proceeds as follows:

(a) In blocking L, the search direction proceeds from the first to the last column; U is blocked from the last to the first column. The following rules will apply to the more critical L blocking; similar rules are used to block U.

(b) The locality of the algorithm is limited to three successive columns, numbered r_a, $r_b = r_a+1$, and $r_c = r_b+1$. If the associated (half-) bandwidths are M_a, M_b, and M_c, then the block is continued in the search direction if $M_b = M_a$. Otherwise, depending on the value of M_c, either a new block is initiated in the column r_b or non-zeros are added to continue the present block. This three-column algorithm

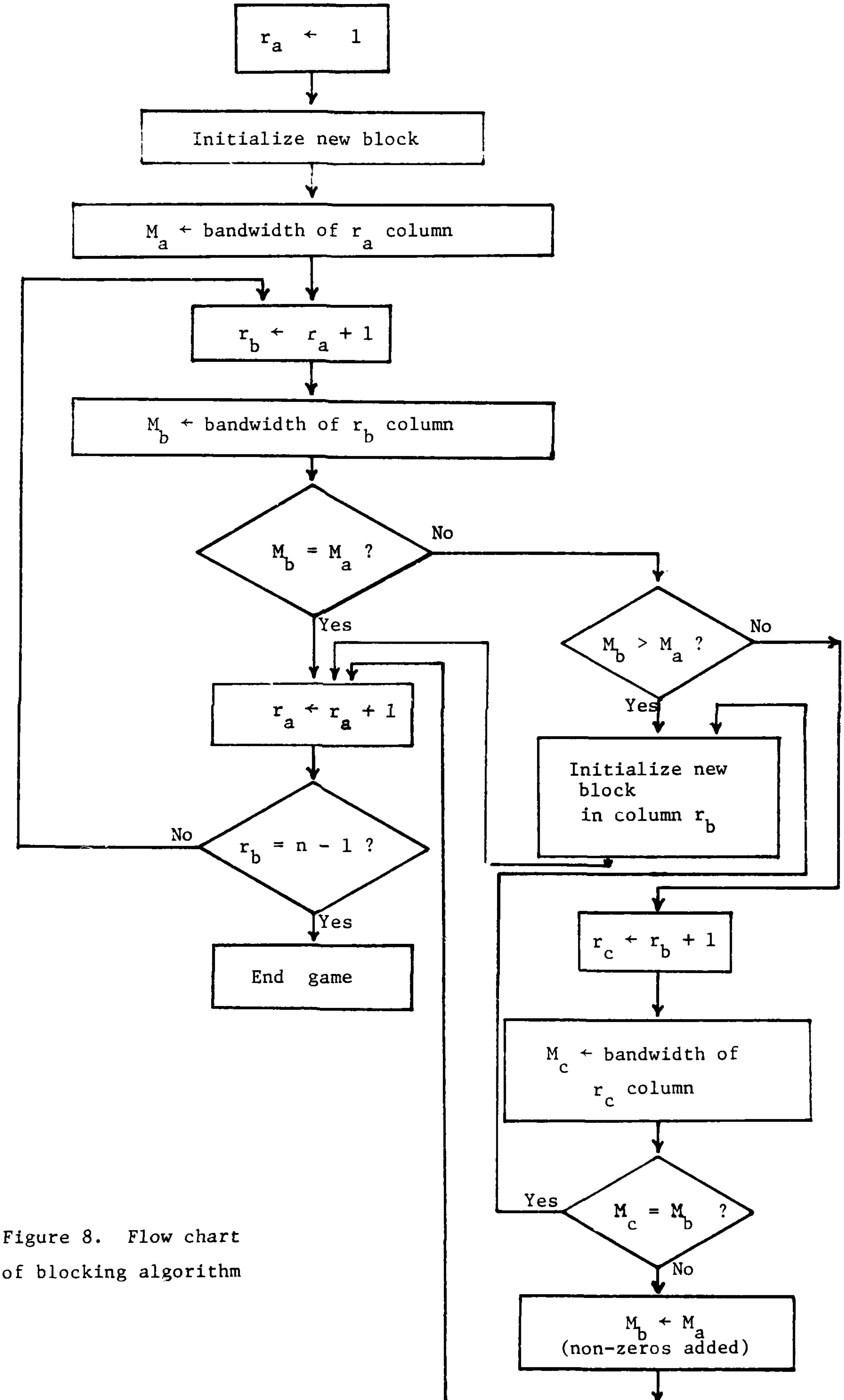


Figure 8. Flow chart of blocking algorithm

has the effect of insuring that a block will be at least two columns wide. The flow-chart for the complete process is given in Figure 8.

(c) The local process of (b) is followed by a more global blocking step where two blocks are merged into a single block if their half-bandwidths differ by less than a preset value.

4. Storage

Although storage is considered a secondary issue to processing speed, storage alternatives deserve consideration before selection of one for implementation.

(a) If repeated solutions are required from the same factorization, then the LU and the matrix (A) storage cannot overlap. If overlapping is permitted, then the LU storage may be allocated ahead of the matrix storage so that, even with fills and inserted non-zeros, the LU storage does not overtake the matrix storage during factorization.

(b) One has the choice of storing columns of each block contiguously, or of interlacing L and U storage in the manner of banded matrix storage. When overlapping LU and the matrix storage is permitted, the choice can become critical. For example, if the matrix has a constant profile so that only one block of U and of L are defined, then contiguous block storage will require that either the entire L or the entire U block be stored ahead of the matrix. In this worst case, total storage will be at least 3/2 of the matrix storage. On the other hand, interlacing columns of L and U must be performed so that each element is directly addressable from a block base address. This will inevitably leave gaps in the LU storage not present in A.

In the blocking algorithm to be discussed, the profile matrix is stored overlapped with A in compressed banded matrix format, i.e., the storage location of logical position (i,j) in LU is (i-j + HBW +1, j), where HBW is the maximum half-bandwidth of L and U. The above-mentioned wasted storage is simply tolerated. The LU storage is allocated in array B at compile time by an EQUIVALENCE statement of the form

```
EQUIVALENCE (A(1), B(L))
```

where the minimum value for L is printed by the blocking subroutine. It is the user's responsibility to ensure that this amount has been set aside before proceeding with the numerical solution. When the symbolic blocking and the numeric solution are carried out in different run steps, this is no problem.

5. Input to Blocking Program

Unsymmetric profile solvers conventionally assume that the data is stored by column or row. Whereas each column (row) is assumed stored compactly, adjacent columns (rows) may not be. To allow such non-compact storage, the following data must be supplied by the user. Note that only storage by column is permitted.

(1) symbolic: the first and last row number in each column;

(2) numeric: the numeric storage location of the diagonal elements.

A characteristic often associated with finite difference grids is the numbering of the nodes without regard for the number of unknowns (U) per grid point. Correspondingly, it is often convenient to input the above profile description assuming U=1 and then have the blocking algorithm expand the description and perform the blocking with U as a parameter.

In summary, only the number of equations, the number of unknowns per grid point, and the descriptors of (1) and (2) are required inputs for the blocking algorithm. The specifics of the software description are contained later in the report.

C. Implementation

The following are major considerations in the code development.

(a) The high-performance kernels of the bandsolver are utilized. Therefore, the performance of the profile solver should approach that of the bandsolver in the special case of a large banded matrix.

(b) In the event blocks are larger than the 64x64 partitions of the bandsolver, the partitioning requirement is imposed during solution. This imposition of both blocking and partitioning strategies accounts for significant overhead in loops above the accumulation kernel. Another source of overhead arises from the provision for re-formatting the user-supplied matrix storage to blocked LU form (see (d) below).

(c) Non-zeroes inserted into U to speedup the back substitution are not processed during the factorization. From Figures 5(b) and 7(b), this tends to affect an irregular grid more than a model grid.

(d) From the timing formula for the accumulation kernel in Equation (8), it may be argued that extending blocks of L to a length of 30 will not increase the aggregate kernel timing. Thus, an L block of length 30 can cover all adjacent blocks of length $\leq$ 30, and the total overhead of processing block descriptors reduced.

D. Performance

1. Timing Evaluation

The common algorithmic measure of MFLOP rate is not an authoritative measure of timing performance for this software since extra computation results from adding non-zeros to produce blocks. Instead, a number of model and irregular grids have been solving using the CRAY-1. Recall that for D4 ordering alternate nodes have been pre-reduced, leaving a profile matrix.

Table 4 gives a timing and storage summary of these runs. In each case, the bandsolver bandwidth was chosen equal to the maximum profile bandwidth. The timing ratios $T_F{:}T_C{:}1$ gives the relative computation times of the Fortran and CAL (respectively) bandsolvers relative to the profile solver. The relative time T_C, with a theoretical upper bound of 2 for a model square grid, is shown to be less than 1 for narrow-band cases, and becomes 1.76 (the largest speedup or profile solution) for large bandwidths. Indeed, Table 5 illustrates the high correlation between speedup and half-bandwidth - whether arising from the profile of a D4-ordered model grid or the natural profile of an irregular grid. The principal exceptions to this monotone behavior are the elongated 32x128 model grid - for which D4 ordering produces a small profile variation - and grid #1 of Figure 6(a), which also has a nearly constant bandwidth. In either case, the overhead of profile solution seems unwarranted.

2. Processor Utilization

For the irregular grids, the floating point computations were counted before and after blocking. Indeed, three counts were made (Table 6).

1. Original Count. This is the count of operations if solution were performed on a scalar processor; this count corresponds to the LU structure in Figure 7(a).

2. Unblocked. The column-by-column elimination requires dense columns of L to the bandedge, represented by the x fill in of L in Figure 7(b). The resulting total operation count is termed the unblocked count.

3. Blocked Count. This count includes the 0 fill in L in Figure 7(b). The factorization count does not include the 0 fill in U, but the (back) substitution count does include this fill.

Model Grids	Grid Nodes	u[1]	HBW[2]	NEQ[3]	CFT Banded Fac.	CFT Banded Sol.	CAL Banded Fac.	CAL Banded Sol.	Profile Fac.	Profile Sol.	Time Ratio Fac.	Time Ratio Sol.	Storage Ratio[4]
16x16	256	1	16	128	6.74	.853	1.45	.162	1.85	.224	3.64:0.78:1	3.81:0.72:1	1.16:1
32x32	1024	1	32	512	60.2	4.32	11.6	.710	11.3	.758	5.32:1.02:1	5.70:0.97:1	1.22:1
64x64	4096	1	64	2048	620.	25.5	141.	4.43	99.7	3.82	6.21:1.41:1	9.98:1.15:1	1.43:1
32x128	4096	1	34	2048	266.	19.2	51.9	2.88	59.1	2.96	4.50:.878:1	6.48:.973:1	1.06:1
32x32	1024	3	98	1536	935.	26.4	263.	5.56	173.	4.46	5.40:1.52:1	5.91:1.24:1	1.42:1
Irreg. Grids													
#1	391	1	7	198	5.31	1.15	1.33	.247	1.91	.271	2.78:0.70:1	4.24:0.91:1	1.07:1
		5	39	990	151.	9.61	30.6	1.52	30.1	1.47	5.02:1.01:1	6.53:1.03:1	1.15:1
		9	71	1782	693.	25.1	213.	6.03	147.	4.57	4.71:1.45:1	5.49:1.31:1	1.14:1
#2	507	1	28	285	27.9	2.26	5.35	.370	5.62	.519	4.96:0.95:1	4.35:0.71:1	1.16:1
		3	86	855	416.	12.9	121.	2.93	75.3	2.43	5.52:1.60:1	5.30:1.20:1	1.36:1
		5	144	1425	1650.	31.7	536.	7.86	304.	6.05	5.42:1.76:1	5.23:1.30:1	1.34:1
#3	2323	1	55	1226	296.	14.0	65.6	2.38	49.5	2.21	5.97:1.32:1	6.33:1.07:1	1.36:1

[1]Unknowns; [2]Maximum half bandwidth; [3]No. of equations; [4]Banded:profile storage

Table 4. Summary of timing results (times in millisec); CFT uses LINPACK codes of SGBFA and SGBSL

Table 6 indicates the largest percentage operation count increase for factorization due to blocking occurs with grid #2. This occurs largely because the L blocks are merged into a single block by the previously-noted strategy of merging all adjacent L blocks of length less than 30. Most blocks of grids #3 are well below the limit.

Of perhaps more significance is the effective MFLOP rate. This rate is obtained by dividing the number of original operations (see above) by the solution time; it therefore discounts the operations due to non-zeros resulting from blocking and allows an intuitive comparison with other vector and with scalar processors. This rate is shown to be over 80 MFLOPS. In contrast, the bandsolver rate is approximately 110 MFLOPS for a half-bandwidth of 55 (see Table 4).

HBW*	Speedup	Grid	u
7	.7	#1(8x69)	1
16	.78	16x16	1
28	.95	#2(23x37)	1
32	1.02	32x32	1
34	.88	32x128	1
39	1.01	#1(8x69)	5
55	1.32	#3(55x72)	1
64	1.41	64x64	1
71	1.45	#1(8x69)	9
86	1.60	#2(23x37)	3
98	1.52	32x32	3
144	1.76	#2(23x37)	5

*Maximum half bandwidth

Table 5. Speedup of profile over banded solution as function of half bandwidth.

NEQ	Original	Unblocked	Blocked
391			
Fac.	15,259	15,469	17,691
Sol.	4,802	4,838	5,322
507			
Fac.	191,917	224,853	281,517
Sol.	19,805	21,429	27,599
2323			
Fac.	3,754,265	3,982,156	4,093,469
Sol.	180,846	187,008	199,190

Table 6. Floating point operations in solution of irregular grids. u = 1.

NEQ	Effective	Actual
391		
Fac.	7.99	9.25
Sol.	17.7	19.6
507		
Fac.	34.1	50.1
Sol.	38.2	53.2
2323		
Fac.	75.8	82.7
Sol.	81.8	90.1

Table 7. Effective and actual MFLOP rates in solution of irregular grids.

E. Software Description and Calling Conventions

1. LU factorization

Call PROFAC (A, N, ICOL, IBL, NBL, IW)

where

A is the matrix array storage

N is the number of equations

ICOL (3*I-2)* is the location in A of the Ith diagonal (pivot position)

ICOL (3*I-1)* is the first (smallest) row number in the Ith column

ICOL (3*I)* is the last (largest) row number in the Ith column

IBL (4*J-3) is the first column number in the Jth block

IBL (4*J-2) is the storage relative to A(1), of the (1,1) position of the Jth block

IBL (4*J-1) is the number of rows in the Jth block (always positive)

IBL (4*J) is the storage increment between the (K,L) and the (K, L+1) positions of the block

NBL is the total number of L and U blocks

IW (I) is the L block number that includes the Ith column

2. Solution (Forward and Backward substitution)

CALL PROSOL (A, N, Y, ICOL, IBL, MOV)

where

Y*contains the right hand side on entry and the solution on exit; it must be dimensioned at least $N + \delta N_1 + \delta N_2$

MOV*is δN_1

δN_1 is 1 - (most negative row number in the blocked U matrix)

δN_2 is (most positive row number in the blocked L matrix) - N

*Input data to subroutine

3. Blocking algorithm

CALL BlOCK (IBL, ICOL, IW, N, M, NPB, MOV, IDSTOR, NBL)

IBL, ICOL*, IW, N*, MOV, NBL are defined in calls to PROFAC and PROSOL

M	is the maximum half-bandwidth
NPB*	is the number of unknowns per grid point
IDSTOR	is the storage used by the matrix A. (M and IDSTOR are used by FORM to formulate example equations, but in general are unnecessary to communicate with PROFAC and PROSOL.)

*Input data to subroutine

F. Example Problems

1. Without Automatic Blocking

A driver program for PROFAC and PROSOL is given in Table 8(a). The symbolic arrays ICOL and IBL need be formed only once; the array IW and the displacement MOV may be formed from these two arrays as shown in the program.

Input data for the profile matrix associated with the D4 ordering of an 8x12 grid (see Figure 5) is given in Table 8(b)-(c). The output array Y(J) has the solution Y(J) = J for the values given.

2. With Automatic Blocking

A driver program (included on the tape with the CAL-coded solver) is listed in Appendix A. For symbolic description, the user may input either (a) ICOL (3*J) and ICOL (3*J-1) or (b) the conventional column-ordered sparse descriptors of L and U, assuming that all columns are dense from the first to the last row number. Numeric storage for the matrix is formed in packed column-ordered format; storage for L and U is in conventional column-ordered banded format, using the maximum half-bandwidth.

```
C******  PROFILE SOLVER DRIVER
      DIMENSION IBL(200),ICOL(300),IW(100)
      DIMENSION A(1000),B(2000),Y(100)
      EQUIVALENCE (A(1),B(1000))
C***   READ SYMBOLIC DATA
      READ(5,1)N,NBL,NA
      N3=3*N
      READ(5,1)(ICOL(J),J=1,N3)
      NBL4=4*NBL
C***   THE LU FACTORS ARE STORED BEGINNING IN NEGATIVE STORAGE
C      LOCATIONS OF MATRIX A STORAGE; THESE ARE INDICATED BY
C      NEGATIVE NUMBERS IN IBL(4*J-3)
      READ(5,1)(IBL(J),J=1,NBL4)
    1 FORMAT(10I5)
      NB=1
      DO 3 J=1,N
      IF(IBL(4*NB-3).EQ.J)NB=NB+1
    3 IW(J)=NB-1
      MOV=0
      DO 4 J=1,NBL
    4 MOV=MAXO(MOV,IABS(IBL(4*J-1)))
C***   READ NUMERIC DATA
      READ(5,2)(A(J),J=1,NA)
      READ(5,2)(Y(J),J=1,N)
    2 FORMAT(10F5.0)
      CALL PROFAC(A,N,ICOL,IBL,NBL,IW)
      CALL PROSOL(A,N,Y,ICOL,IBL,MOV)
      WRITE(6,5)(Y(J),J=1,N)
    5 FORMAT(5E12.4)
      STOP
      END
```

(a) Driver program

```
   48   16  732
    1    1    5    7    1    6   14    1    9   24
    1   10   35    1   11   46    2   12   57    3
   15   71    3   16   86    3   17  101    4   18
  116    5   19  131    6   20  146    7   23  164
    7   24  183    7   25  202    8   26  221    9
   27  240   10   28  259   11   28  277   12   28
  294   13   31  314   13   32  335   13   33  356
   14   34  377   15   35  398   16   36  419   17
   36  439   18   36  456   21   37  474   21   38
  493   21   39  512   22   40  531   23   41  550
   24   42  569   25   42  587   26   42  602   29
   43  617   30   44  632   31   45  647   32   46
  662   33   46  676   34   46  687   37   47  698
   38   48  709   39   48  719   40   48  726   43
   48  732   44   48
    1 -306    4   21    3 -264    6   21    7 -180
    8   21   13  -54   10   21   28  261    8   21
   36  429    6   21   42  555    4   21   48  681
    0   21   48  680    5  -21   46  638    7  -21
   42  554    9  -21   36  428   11  -21   21  113
    9  -21   13  -55    7  -21    7 -181    5  -21
    1 -307    1  -21
```

(b) Symbolic input data

Table 8. Sample driver program and input data

20.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	20.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.
-1.	-1.	-1.	-1.	-1.	-1.	-1.	-1.	20.	-1.
-1.	-1.	-1.	-1.	-1.	20.	-1.	-1.	-1.	-1.
-1.	20.								
6.	21.	18.	29.	39.	49.	30.	35.	39.	45.
51.	57.	18.	15.	11.	13.	15.	17.	48.	80.
23.	12.	-0.	-0.	-0.	-0.	37.	75.	116.	99.
81.	83.	85.	87.	132.	178.	237.	243.	249.	255.
308.	362.	441.	451.	510.	570.	714.	778.		

(b) Numeric input data
Table 8. Continued

References

[1] Jordan, T., and K. Fong, "Some Linear Algebraic Algorithms and and Their Performance on the CRAY-1," Report LA-6774, Los Alamos National Laboratory, June, 1977.

[2] Calahan, D. A., "A Block-Oriented Equation Solver for the CRAY-1," Report #136, Systems Engineering Laboratory, University of Michigan, Ann Arbor, December 1, 1980.

[3] Orbits, D. A., "A CRAY-1 Simulator," Report #118, Systems Engineering Laboratory, University of Michigan, Ann Arbor, September, 1978.

[4] Duff, I. S., and J. K. Reid, "Experience of Sparse Matrix Codes on the CRAY-1," Report CSS 116, AERE Harwell, October, 1981.

[5] George, J. A., "Nested Dissection of a Regular Finite Element Mesh," Siam Jour. Num. Anal., vol. 10, 1973, pp. 345-363.

[6] Calahan, D. A., and W. G. Ames, "Vector Processors: Models and Applications," Trans. IEEE, vol. CAS-26, no. 9, September, 1979, pp. 715-726.

[7] Calahan, D.A., "Performance of Linear Algebra Codes on the CRAY-1," SPE Journal, October, 1981, pp. 558-564.

[8] Gustavson, F. G., "Some Basic Techniques for Solving Sparse Systems of Linear Equations," in Sparse Matrices and Their Applications, Ed. by Rose and Willoughby, Plenum Press, 1972, pp. 41-52.

[9] Calahan, D. A., P. G. Buning, and W. N. Joy, "Vectorized General Sparsity Algorithms with Backing Store," Report #96, Systems Engineering Laboratory, University of Michigan, January 1977.

[10] Calahan, D. A., "Sparse Vectorized Direct Solution of Elliptic Problems," in Elliptic Problem Solvers, Ed. by M. H. Schultz, Academic Press, 1981, pp. 241-245.

[11] Price, H.S. and K. H. Coates, "Direct Methods in Reservoir Simulation," SPE Journal, vol. 14, 1974, pp. 295.308.

[12] Woo, P. T., and J. M. Levesque, "Benchmarking a Sparse Elimination Routine on the CYBER 205 and the CRAY-1," 6th SPE Symposium on Reservoir Simulation, New Orleans, Feb. 1-3, 1982, pp. 535-538.

[13] Calahan, D.A., W. G. Ames, and E. J. Sesek, "A Collection of Equation-Solving Codes for the CRAY-1," Systems engineering Laboratory, University of Michigan, August 1, 1979, (rev. 11/71).

[14] Woo, P. T., S. C. Eisenstat, M. H. Schultz, and A. H. Sherman, "Application of Sparse Matrix Techniques to Reservoir Simulation," in Sparse Matrix Computations, Ed. by Bunch and Rose, Academic Press, 1976, pp. 527-438.

APPENDIX A

Listing of Blocking Program

```
C****  PROFILE BLOCKING AND SOLUTION DRIVER
C****  IBL HAS DIMENSION .GE. 4*( # OF BLOCKS IN L & U COMBINED )
C      ICOL HAS DIMENSION .GE. 3*( # OF EQUATIONS )
C      IW AND IT HAVE DIMENSION .GE. # OF EQUATIONS
C      Y HAS DIMENSION .GE. # OF EQUATIONS + 2*MOV, WHERE MOV
C        IS THE DISTANCE THE RHS IS SHIFTED IN PROSOL BEFORE AND
C        AFTER FORWARD AND BACK SUBSTITUTION; MOV .GE. MAXIMUM
C        BANDWIDTH FOR SAFETY
C      A STORES THE MATRIX IN A MANNER CONSISTENT WITH ICOL
C      B STORES THE L AND U FACTORS; ITS STORAGE IS OVERLAPPED
C        WITH A SO THAT (1) NEGATIVE ADDRESSES IN A ARE POSITIVE
C        ADDRESSES IN B, AND (2) AS L AND U FACTORS ARE FORMED
C        IN B THEY DO NOT OVERTAKE THE MATRIX ELEMENTS STORED
C        IN A
C****  SUBROUTINES LOCPIV AND FORM MUST BE CHANGED
C        CONSISTENTLY IF MATRIX NUMERIC STORAGE IS NOT COMPACTED
C        AND IN COLUMN ORDER
      DIMENSION IBL(30000), ICOL(27000), IW(9000), IT(9000)          1
      DIMENSION Y(10000), A(200000), B(300000)                       2
      DIMENSION IA(20000),JA(600)                                    3
      EQUIVALENCE (A,IA),(IW,JA)                                     4
      EQUIVALENCE (A(1), B(100000))                                  5
   10 CONTINUE                                                       6
C****  CHOOSE EITHER D4 OR READLU FOR INPUT ROUTINE
C     CALL D4(ICOL, N, NPB)
      CALL READLU(ICOL,IA,JA,N,NPB)                                  7
C****  SYMBOLIC PROCESSING; EXECUTE ONCE FOR FIXED MATRIX STRUCTURE
      CALL BLOCK(IBL, ICOL, IW, N, M, NPB, MOV, IDSTOR, NBL, NBTOT)  8
C****  NUMERIC PROCESSING; THE FOLLOWING MAY REPRESENT A SECOND
C        PROGRAM THAT PERFORMS REPEATED SOLUTIONS
      CALL FORM(Y, A, ICOL, N, IDSTOR, M)                            9
      T1=SECOND()                                                   10
      CALL PROFAC(A, N, ICOL, IBL, NBTOT, IW)                       11
      T1=SECOND()-T1                                                12
      T2=SECOND()                                                   13
      CALL PROSOL(A, N, Y, ICOL, IBL, MOV)                          14
      T2=SECOND()-T2                                                15
      WRITE (6,96) T1,T2                                            16
   96 FORMAT(2E15.7)                                                17
C****  CHECK ANSWER
      DO 20 I = 1, N                                                18
        AI = I                                                      19
        IF (ABS(Y(I) - AI) .GT. 1.E-6) GO TO 30                     20
   20 CONTINUE                                                      21
      STOP                                                          22
   30 WRITE (6,40) I                                                23
   40 FORMAT (' FIRST ERROR IN', I6, ' VARIABLE')                   24
      STOP                                                          25
      END                                                           26
```

```
                                                                                  ISN
      SUBROUTINE D4(ICOL, N, NPB)                                                  1
C***   DETERMINES LAST & 1ST ROW NUMBERS FOR PROFILE MATRIX
C        RESULTING FROM D4 ORDERING OF MXN GRID; M & N EVEN
      DIMENSION ICOL(1)                                                            2
C***   N INITIALLY IS SMALLER GRID DIMENSION
C      N IS # OF EQUATIONS ON EXIT
C      M IS LARGER GRID DIMENSION
C      NPB IS # OF UNKNOWNS PER GRID POINT
      READ (5,10) M, N, NPB                                                        3
   10 FORMAT (3I5)                                                                 4
      IF (NPB .EQ. 0) NPB = 1                                                      5
C***   LAST ROW #
      NM = N * M / 2                                                               6
      NSW1 = (N/2) * (N/2 - 1) + 1                                                 7
      NSW2 = NM - (N/2) * (N/2 + 1)                                                8
      L1 = 1                                                                       9
      L2 = N / 2 - 1                                                              10
      K = 5                                                                       11
      ICOL(3) = K                                                                 12
      JSW1 = 3                                                                    13
      JSW2 = NSW1 + N - 1                                                         14
      DO 70 J = 2, NM                                                             15
        IF (J .GT. NSW2) GO TO 50                                                 16
        IF (J .NE. JSW1) GO TO 30                                                 17
        IF (J .GT. NSW1) GO TO 20                                                 18
        L1 = L1 + 1                                                               19
        JSW1 = L1 * (L1 + 1) + 1                                                  20
        K = K + 2                                                                 21
        GO TO 60                                                                  22
   20   JSW1 = JSW1 + N                                                           23
        K = K + 2                                                                 24
   30   IF (J .NE. JSW2) GO TO 60                                                 25
        JSW2 = JSW2 + N                                                           26
   40   IF (J .LT. NM - 4) ICOL(3*(J - 1)) = ICOL(3*(J - 1)) - 1                  27
        K = K - 2                                                                 28
        GO TO 60                                                                  29
   50   IF (J .NE. JSW2) GO TO 60                                                 30
        L2 = L2 - 1                                                               31
        JSW2 = NM - L2 * (L2 + 1)                                                 32
        GO TO 40                                                                  33
   60   K = K + 1                                                                 34
   70 ICOL(3*J) = MINO(K,NM)                                                      35
C***   FIRST ROW #
      JSW1 = NSW1                                                                 36
      NSW1 = NM - NSW2 + 1                                                        37
      NSW2 = NM - JSW1 + 1                                                        38
      JSW1 = 2                                                                    39
      L1 = 1                                                                      40
      L2 = N / 2 - 1                                                              41
      IF (M .EQ. N) L2 = L2 - 1                                                   42
      JSW2 = MINO(NM - L2*(L2 + 1) + 1,NSW1 + N)                                  43
      K = -1                                                                      44
      ICOL(2) = 1                                                                 45
      DO 140 J = 2, NM                                                            46
        IF (J .GT. NSW2) GO TO 120                                                47
        IF (J .NE. JSW1) GO TO 100                                                48
        IF (J .GT. NSW1) GO TO 90                                                 49
        L1 = L1 + 1                                                               50
```

ISN

```
      JSW1 = L1 * (L1 + 1) + 2
   80 ICOL(3*J - 1) = ICOL(3*(J - 1) - 1)
      K = K - 1
      GO TO 140
   90 JSW1 = JSW1 + N
      GO TO 80
  100 IF (J .NE. JSW2) GO TO 130
      JSW2 = JSW2 + N
  110 K = K + 2
      GO TO 130
  120 IF (J .NE. JSW2) GO TO 130
      L2 = L2 - 1
      JSW2 = NM - L2 * (L2 + 1) + 1
      GO TO 110
  130 K = K + 1
      ICOL(3*J - 1) = MAXO(1,K)
  140 CONTINUE
      N1 = N
      N = N * M / 2
      IF (N1 .NE. M) M = N1 + 2
      RETURN
      END
```

ISN

```
C****  THIS ROUTINE SYMBOLICALLY MERGES ADJACENT BLOCKS OF
C        L&U WITH HALF-BANDWIDTHS THAT DIFFER BY .LE. IDEL
      SUBROUTINE BLKCMP(IBL, I1, I2, I3)
      DIMENSION IBL(1)
      IDEL = 0
      I1P1 = I1 + 1
      IF (I1P1 .GT. I2) GO TO 40
      DO 30 I = I1P1, I2
   10   LEN1 = IBL(4*I - 1)
        LEN2 = IBL(4*(I - 1) - 1)
        IF (IABS(LEN1 - LEN2) .GT. IDEL) GO TO 30
        LEN1 = MAXO(LEN1,LEN2)
        I3M1 = I3 - 1
        IF (I .GT. I3M1) GO TO 30
        DO 20 J = I, I3M1
        DO 20 K = 1, 4
   20   IBL(4*J - K + 1) = IBL(4*(J + 1) - K + 1)
        I3 = I3M1
        I2 = I2 - 1
        GO TO 10
   30 CONTINUE
   40 RETURN
      END
```

```
C      SUBROUTINE READLU(ICOL, N, NPB)
C      DIMENSION ICOL(1)
C      READ (5,10) N, NPB
C   10 FORMAT (16I5)
C      N3 = 3 * N
C      READ (5,10) (ICOL(J),J=1,N3)
C      RETURN
C      END
C***   ALTERNATE READ OF LU MATRIX IN COLUMN-ORDERED,SPARSE
C***   FORMAT; NOTE ARGUMENT LIST IS DIFFERENT.
       SUBROUTINE READLU(ICOL,IA,JA,N,NPB)
       DIMENSION ICOL(1),IA(1),JA(1)
       READ(5,10) N
    10 FORMAT(16I5)
       NP1=N+1
       READ(5,10)(JA(J),J=1,NP1)
       NA=JA(NP1)-1
       READ(5,10)(IA(J),J=1,NA)
       DO 1 J=1,N
       ICOL(3*J-1)=IA(JA(J))
     1 ICOL(3*J)=IA(JA(J+1)-1)
       NPB=1
       RETURN
       END
```

```
C***   DETERMINE NUMERIC LOCATION IC(3*J-2) OF JTH DIAGONAL ELEMENT
C       IN A, IF NOT GIVEN IN INPUT DATA; ASSUME MATRIX NUMERIC STORAGE
C       IS TO BE PACKED BY COLUMN, AS GIVEN IN SUBROUTINE FORM
      SUBROUTINE LOCPIV(ICOL, N, NPB, IDSTOR)
      DIMENSION ICOL(1)
C****  IDSTOR IS STORAGE LOCATION OF LAST POSITION OF MATRIX (ANN)
      IDSTOR = 0
      DO 10 J = 1, N
        IC1 = ICOL(3*J - 1)
        IC2 = ICOL(3*J)
        DO 10 L = 1, NPB
        IDSTOR = IDSTOR + NPB * (J - IC1) + L
        ICOL(3*L + 3*NPB*(J - 1) - 2) = IDSTOR
   10 IDSTOR = IDSTOR + (IC2 - J) * NPB - (L - 1) + NPB - 1
      RETURN
      END
```

```
                                                                                          ISN
C****  THIS BLOCKING ROUTINE ASSUMES L AND U ARE NOT TO BE
C        COMPACTED BY COLUMN; RATHER, A CONVENTIONAL COMPRESSED
C        BANDED STORAGE SCHEME IS ASSUMED, I.E., AS A 2-D ARRAY
C        OF DIMENSION (2*MAXBW+1,N), WHERE MAXBW IS MAX. HALF
C        BANDWIDTH.  THIS WASTES SPACE BETWEEN THE L&U PROFILE AND .
C        THE BANDEDGE
      SUBROUTINE BLOCK(IBL, ICOL, IW, N, M, NPB, MOV, IDSTOR, NBL,                          1
     1        NBTOT)
      DIMENSION IW(1), IBL(1), ICOL(1)                                                      2
C****  FIND MAXIMUM BANDWIDTH
      MAXBW = 0                                                                             3
      DO 10 J = 1, N                                                                        4
        MAXBW = MAXO(MAXBW,(ICOL(3*J) - J + 1)*NPB - 1)                                     5
   10 MAXBW = MAXO(MAXBW,(J - ICOL(3*J - 1) + 1)*NPB - 1)                                   6
      MOV = MAXBW                                                                           7
      M = ((MOV+1)/NPB)-1                                                                   8
C***  ALL BLOCKS OF L WILL HAVE A MINIMUM BANDWIDTH OF MX
      MX = 0                                                                                9
      M2 = 2 * M + 1                                                                       10
      MP1 = M + 1                                                                          11
      MP2 = MP1 + 1                                                                        12
C***   LOCATE PIVOT STORAGE
      CALL LOCPIV(ICOL, N, NPB, IDSTOR)                                                    13
C****  DETERMINE (1) OFFSET BETWEEN MATRIX & L&U STORAGE
C             (2) RHS STORAGE REQUIREMENTS
      IOFF = (NPB**2) * (2*M + 1) * (N + 2) - IDSTOR                                       14
      NBL = N + 2 * MOV                                                                    15
      WRITE (6,20) IOFF, NBL                                                               16
   20 FORMAT (' EQUIVALENCE (A(1),B(L)), WHERE L .GE.', I6/                                17
     1       ' DIMENSION RHS .GE.', I6)
      NBL = 0                                                                              18
      IFLAG = 0                                                                            19
      J = 1                                                                                20
      GO TO 60                                                                             21
C***   BLOCK L
   30 J = J + 1                                                                            22
      IF (ICOL(3*J) - ICOL(3*(J - 1)) - 1) 40, 50, 60                                      23
   40 IFLAG = 1                                                                            24
      GO TO 70                                                                             25
   50 IF (IFLAG .EQ. 0) GO TO 70                                                           26
      IFLAG = 0                                                                            27
      J = J - 1                                                                            28
   60 NBL = NBL + 1                                                                        29
      N4 = 4 * NBL                                                                         30
      IBL(N4 - 3) = 1 + NPB * (J - 1)                                                      31
      IBL(N4 - 2) = -IOFF + MP1 * NPB + 1 + (J - 1) * NPB * (2*(NPB*M +                    32
     1NPB - 1) + 1)
      IBL(N4 - 1) = MAXO((ICOL(3*J) - J)*NPB + NPB - 1,MX)                                 33
      IBL(N4) = M2 * NPB + NPB - 1                                                         34
      MOV = MAXO(MOV,IBL(N4 - 1))                                                          35
   70 IF (J .NE. N - 1) GO TO 80                                                           36
C***   LAST L BLOCK
      J = J + 1                                                                            37
      NBL = NBL + 1                                                                        38
      N4 = 4 * NBL                                                                         39
      IBL(N4 - 3) = N * NPB                                                                40
      IBL(N4 - 2) = -IOFF + N * NPB * (2*(NPB*M + NPB - 1) + 1) - M *                      41
     1NPB + 2 - NPB
```

```
                                                                          ISN
      IBL(N4 - 1) = 0                                                      42
      IBL(N4) = -M2 * NPB + NPB + 1                                        43
   80 IF (J .LT. N) GO TO 30                                               44
      NBLB = 0                                                             45
      IFLAG = 0                                                            46
      GO TO 120                                                            47
C***   BLOCK U
   90 J = J - 1                                                            48
      IF (ICOL(3*(J + 1) - 1) - ICOL(3*J - 1) - 1) 100, 110, 120          49
  100 IFLAG = 1                                                            50
      GO TO 130                                                            51
  110 IF (IFLAG .EQ. 0) GO TO 130                                          52
      IFLAG = 0                                                            53
      J = J + 1                                                            54
  120 NBLB = NBLB + 1                                                      55
      N4 = 4 * NBL + 4 * NBLB                                              56
      IBL(N4 - 3) = J * NPB                                                57
      IBL(N4 - 2) = -IOFF + J * NPB * (2*(NPB*M + NPB - 1) + 1) - NPB *    58
     1M + 1 - NPB
      IBL(N4 - 1) = (J - ICOL(3*J - 1) + 1) * NPB                          59
      IBL(N4) = -M2 * NPB - NPB + 1                                        60
      MOV = MAXO(MOV,IBL(N4 - 1) - NPB)                                    61
  130 IF (J .NE. 2) GO TO 140                                              62
C***   LAST U BLOCK
      J = J - 1                                                            63
      NBLB = NBLB + 1                                                      64
      N4 = N4 + 4                                                          65
      IBL(N4 - 3) = 1                                                      66
      IBL(N4 - 2) = -IOFF + MP1 * NPB                                      67
      IBL(N4 - 1) = 1                                                      68
      IBL(N4) = -M2 * NPB - NPB + 1                                        69
  140 IF (J .GT. 1) GO TO 90                                               70
      NBTOT = NBL + NBLB                                                   71
      CALL BLKCMP(IBL, 1, NBL, NBTOT)                                      72
      NBLP1 = NBL + 1                                                      73
      CALL BLKCMP(IBL, NBLP1, NBTOT, NBTOT)                                74
      IBL(4*NBTOT + 1) = 0                                                 75
      N = N * NPB                                                          76
      NB = 1                                                               77
      DO 150 J = 1, N                                                      78
        IF (IBL(4*NB - 3) .EQ. J) NB = NB + 1                              79
  150 IW(J) = NB - 1                                                       80
      CALL NEWCOL(N, NPB, ICOL)                                            81
      M=(M+1)*NPB-1                                                        82
      RETURN                                                               83
      END                                                                  84
```

```
C****  FORMULATE NUMERIC VALUES IN PACKED COLUMN-ORDERED ARRAY A
C      OFF-DIAGONALS = -1.; DIAGONALS = 2*M+1.
C      RHS VALUES CHOSEN SO THAT SOLUTION WILL BE Y(J)=J
C****  NOTE THAT MATRIX CAN BE STORED IN A IN ANY MANNER
C      CONSISTENT WITH ICOL; THIS DENSE PACKING IS ONLY ONE
C      POSSIBILITY
      SUBROUTINE FORM(Y, A, ICOL, N, IDSTOR, M)
      DIMENSION Y(1), A(1), ICOL(1)
      DO 10 J = 1, N
   10 Y(J) = 0
      DO 20 I = 1, IDSTOR
   20 A(I) = 0.
      IX = 0
      DO 40 I = 1, N
        ID = ICOL(3*I - 2)
        IF = ICOL(3*I - 1)
        IL = ICOL(3*I)
        DO 30 J = IF, IL
        IX = IX + 1
        A(IX) = -1.
   30   Y(J) = Y(J) - I
        A(ID) = 2 * M
   40 Y(I) = Y(I) + (2*M + 1) * I
      RETURN
      END
```

```
      SUBROUTINE NEWCOL(N, NPB, ICOL)
      DIMENSION ICOL(1)
C***   GENERATE NEW ROW #'S WHEN NPB .NE. 1
      DO 10 I = 1, N
        II = N - I + 1
        DO 10 LL = 1, NPB
        L = NPB - LL + 1
        N3 = 3 + 3 * NPB * (II - 1) + 3 * (L - 1)
        ICOL(N3) = NPB * ICOL(3*II)
   10 ICOL(N3 - 1) = NPB * ICOL(3*II - 1) - NPB + 1
      RETURN
      END
```

3-8

DT